BULLETIN D'INFORMATIQUE APPROFONDIE ET APPLICATIONS

COMPUTATION - INFORMATION

N° 67 - MARS 2004

http://scamup.univ-mrs.fr/biaa

Publication trimestrielle, gratuite, de l'Université de Provence

Dépôt légal : janvier 2004

ISSN 0291 - 5413

BULLETIN D'INFORMATIQUE APPROFONDIE ET APPLICATIONS

COMPUTATION - INFORMATION

N° 67 – MARS 2004

Université de Provence
Equipe Hermès. Case 33
3, place Victor Hugo
F - 13331 Marseille Cedex 3
Téléphone: (0)4 91 10 62 30
Télécopie : (0)4 91 50 91 10

DEPOSITAIRE :

Université de Provence
Bibliothèque Vniversitaire
1, place Victor Hugo
F - 13331 Marseille Cedex 3
Téléphone: (0)4 91 10 85 29
Télécopie : (0)4 91 95 75 57

IMPRIMEUR :

Université de Provence
Service Reprographie
3, place Victor Hugo
F - 13331 Marseille Cedex 3
Téléphone: (0)4 91 10 60 48

http://scamup.univ-mrs.fr/biaa

Publication trimestrielle, gratuite, de l'Université de Provence

Impression : juin 2005

ISSN 0291 - 5413

ÉDITORIAL

Informatique et politique

Edmond Bianco

Le métier d'épicier est un métier noble, comme tous les métiers utiles, et la mise à disposition de tout le monde, de manière directement accessible, de produits aussi importants que ceux qui composent la nourriture de base, est une fonction de service public fondamentale. Mais hélas, le principe de Peters vient également faire des ravages dans ce domaine.

Depuis quelques décennies le petit épicier a pris de l'ambition, il s'est élevé jusqu'aux rênes de l'Etat. Perdus dans la masse des épiciers honnêtes, on trouve toujours quelques médiocres qui compensent leur médiocrité par une ambition démesurée. Et on peut en admirer désormais quelque beau spécimen, bossu à force de calculer sur des bouts de chandelles, avec un crayon qu'il glisse au repos sur l'oreille droite, tandis que sur l'oreille gauche palpite une cigarette en attente.

Mais notre candidat à la gloire posthume et au sacrifice ostensible (ostentatoire ?) se doit devant les écrans de télévision d'exprimer l'image de la force invincible, on le voit le front bas, on imagine très bien la corne vibrante, l'ensemble évoque le taureau prêt à foncer contre les ennemis de la France d'en bas. Le puissant crochet du droit souligne l'image du battant. Toutefois, ce genre de personnage sait bien choisir son petit personnel, c'est atavique. Pour atténuer la force brutale du personnage, un peu de charme, et du charme exotique est indispensable pour séduire les forces cérébrales de la nation. La pauvre Claudie, toute droite tombée des cieux dans ses bottes de cosmonaute se doit d'affrioler, d'aguicher les grincheux du savoir qui voudraient creuser toujours plus profondément dans des domaines du savoir, dont ils devraient bien comprendre que d'abord leur quête est aléatoire, et que, de toute manière, leurs produits ne s'exposent pas sur une étagère d'épicerie. L'ancien épicier-chef, dit " le Mammouth ", avait déjà aiguillé la formation plutôt générale, à la française, vers une autre beaucoup plus rentable pour les industriels qui voisinent une école, et n'ont que faire de gens parlant latin ou grec, alors que savoir coudre ou coller du plastique est plus performant et plus rapide d'acquisition pour le montage des chaussures de sport, ou toute autre fabrication de haut niveau qui se vend sur les marchés des foires. Ne faut-il pas lutter contre la délocalisation ? Autant le faire avec les mêmes armes. N'oublions pas que ce qui coûte cher c'est la main d'œuvre. Aussi préparons-nous à fabriquer de la bonne main d'œuvre à bon marché à qui on aura au préalable ôté toute velléité d'accès à une improbable culture, par ailleurs tout à fait inutile pour concurrencer les petits enfants hindous, ou les va-nu-pieds du " quint " monde.

La société évolue, il est normal qu'une société d'anciens petits boutiquiers soit dirigée par un grand satrape. La fine étroitesse de la profession, " un sous c'est un sou n'est ce pas ? ", se doit d'être compensée par l'ouverture de petits fastes entre amis de même classe. La poussière surannée de l'arrière boutique oblige quelque peu à se détendre de temps en temps en des lupanars au luxe de clinquant. " Peep-show " privés et " eyes wide shut " à usage de châteaux réservés. Encore doutes-je que le grand Satrape invite souvent à sa table le petit bossu de la rue Quinquempoix.

Autant qu'il existe une France d'en Haut, et une France d'en Bas, commence à émerger une autre subdivision de cette malheureuse nation, la France des épiciers et la France des intouchables, cette dernière étant par essence, taillable et corvéable à merci.

Il me paraissait important, travaillant dans cette branche, l'informatique, au développement si rapide et à l'importance désormais tellement fondamentale, de réfléchir à l'impact de ses applications sur la société, et d'en discuter les conséquences.

On pouvait dire, par exemple, que les choix politiques à son propos n'avaient pas été les meilleurs. Les politiciens au pouvoir n'avaient pas compris grand-chose aux immenses possibilités de ce prolongement appliqué des mathématiques, conseillés qu'ils avaient été par des ratés de la profession. On pouvait dire que, définie au départ comme une " big science " par certains et comme un ornement publicitaire par d'autres, on a laissé la richesse naturelle de développement tomber rapidement entre les mains des marchands de tapis et de soupe en gros. Il y a peu encore on pouvait exprimer des avis, amorces éventuelles de discussions susceptibles d'intéresser des gens plus liés à l'intérêt général qu'à l'engraissement des holdings d'outre Atlantique qui verrouillent le développement de l'informatique. On pouvait encore regretter ceci ou pousser au développement de cela ...

Désormais, ce qui dépasse un peu l'importance du prix du papier de toilette, où l'innovation en terme de contrôle de la vitesse sur les routes, voire le calcul du maximum de surface que doit comporter en millimètres carrés, le voile des jeunes pucelles musulmanes, a peu de chances d'accéder à l'intellect des maîtres de l'épicerie France. Tant pis.

En attendant des jours meilleurs, nous parlerons donc d'autre chose.

De Jacques Arsac à Charles Duchâteau
Images pour programmer

Jean - Michel Knippel

knippel@up.univ-mrs.fr

Dans les années 1970, je commençais mon chemin universitaire et je me souviens avoir croisé les graphes de Nassi-Shneiderman au cours d'un exposé de stage de la filière Automatique-Informatique à Marseille. Personne, ne m'avait parlé de cela au cours de mes études informatiques à Lyon. J'en arrive, une trentaine d'années plus tard, à la conclusion que ceci est une affaire de culture, non pas informatique, mais de culture géographique.

Lors des années 1980, l'université me confia un cours d'algorithmique, je profitais de cette occasion pour intégrer les graphes de Nassi-Shneiderman dans mon enseignement. Nous en reparlerons plus en détail dans le numéro suivant du bulletin.
Je reviens sur cet aspect culturel, je ne trouvais alors que fort peu de livres en français, intégrant ces graphes : en Belgique, au Luxembourg et en Suisse. Que se passe t'il donc en France ?

Dans les années 1990, je lisais l'ouvrage épuisé et réédité sous forme de syllabus :
Charles Duchâteau préfacé par Jacques Arsac
Images pour programmer. Apprendre les concepts de base
De Boeck-Wesmael. Editions Universitaires.

Tout naturellement, je le recommandais à mes étudiants et étudiantes et je prenais contact avec mon collègue de l'Institut d'Informatique de Namur des facultés universitaires Notre-Dame de la Paix en Belgique. Je lui proposais de diffuser quelques parties de ses supports d'enseignement d'introduction à l'algorithmique. Ceci se fait dans la collection 2004 de notre périodique. Le lecteur ou la lectrice qui voudraient aller plus loin pourront consulter les supports de Charles Duchâteau au format papier ou en ligne [*], [**].

Les objectifs de l'enseignement sont les suivants :
- faire découvrir les concepts et principes de base de l'algorithmique et de la programmation impérative ;
- faire comprendre comment ces concepts s'incarnent dans le langage Pascal, maîtriser suffisamment la syntaxe de pascal ;
- rendre capable de mettre en oeuvre ces concepts, à travers une démarche descendante, pour analyser des tâches simples et écrire les programmes correspondants en Pascal.

Jacques Arsac ? Je ne l'ai pas oublié. Professeur émérite de l'Université Pierre et Marie Curie et membre de l'Académie des Sciences, il a préfacé le livre insistant sur les images offertes à l'apprenti programmeur, images qu'il devra dépasser.

Nous retrouvons l'informatique et les rapports forme et sens. Jacques Arsac a développé ces thèmes dans un séminaire de l'IRIN à Nantes en novembre 2002.

Quelques références de Charles Duchâteau dans l'article suivant

[3] ARSAC J.
Premières leçons de programmation
Cedic/Nathan. Paris. 1980

[6] ARSAC J.
Les bases de la programmation
Dunod. Paris. 1983

[7] ARSAC J.
Les machines à penser. Des ordinateurs et des hommes
Editions du Seuil. Paris. 1987

[15] BIONDI J., CLAVEL G.
Introduction à la programmation 1. Algorithmique et langage.
Masson. Paris. 1984

[*] DUCHATEAU C.
Images pour programmer. Apprendre les concepts de base (Vol. 1)
CeFIS. FUNDP. Namur. 2002
http://www.det.fundp.ac.be/cefis/publications/charles/images1-5-79.pdf

[**] DUCHATEAU C.
Images pour programmer. Tableaux et approche descendante (Vol. 2)
CeFIS. Réf. 5.78. FUNDP. Namur. 2002
http://www.det.fundp.ac.be/cefis/publications/charles/images2-5-78.pdf

[43] JAMES M.
Elegant Programming
Computing Today. Août 1982. pp. 24-37

[51] LEDGARD H.F.
Proverbes de programmation
Dunod. Paris. 1975

5

Images pour programmer : programmer !

Charles Duchâteau

Si j'écris le mot "informatique", nul doute que vous pensiez d'abord "ordinateur". Si j'avais écrit "météorologie", les images associées auraient probablement été "temps, anticyclone, dépression, pluie, ..." et pour certains ... "thermomètre". Le mot "astronomie" aurait, lui, appelé "étoiles, planètes, lune, ..." et peut-être "télescope". Mais personne n'est prêt à dire que la météorologie est la science des thermomètres ou l'astronomie celle des télescopes.[1] Et bien, l'informatique n'est pas plus la science des ordinateurs que l'astronomie n'est la science des télescopes ou la météorologie celle des thermomètres ! L'ordinateur est un outil essentiel de l'informatique, comme le télescope est un outil pour l'astronome, mais "faire de l'informatique", c'est bien plus (et bien plus passionnant) que connaître et manipuler un ordinateur.

Cet ouvrage montrera, je l'espère, que les problèmes sont ailleurs que dans la connaissance et la maîtrise d'un ordinateur : j'aborderai fort peu les aspects techniques et dirai bien peu de choses de l'intérieur de l'ordinateur. Pourtant, même s'il sera rejeté à la périphérie de nos préoccupations, il me faut dire un mot de ce qu'il est, ne serait-ce que pour préciser, parmi les multiples facettes qui pourraient être retenues, celle sur laquelle je placerai un éclairage particulier.

1. Un ordinateur, c'est quoi ?[2]

1.1 Une définition ?

Je ne tomberai pas dans le piège consistant à proposer une définition précise ou exhaustive de ce qu'est un ordinateur; parmi la multitude des choses qu'on pourrait en dire, je me contente de souligner celles que, pour mon propos, je souhaite placer en avant :

> ***Un ordinateur, c'est une machine à traiter des informations, de manière formelle*, pour autant qu'on lui ait indiqué (dans le détail !) comment mener à bien ce traitement.**

S'il s'agissait là d'une définition, il resterait à définir "machine", "information", "traitement",... Il faudrait encore ajouter que cette machine est électrique, qu'elle est constituée de circuits électroniques, ... ce qui n'éclairerait en rien mon propos.

---

1 Cf. [6]

2 Le lecteur intéressé pourra consulter les chapitres 2 et 3 de "Initiation à l'informatique" à l'adresse http://www.det.fundp.ac.be/cefis/publications/charles/ini-5-51.pdf.

1.2 Traitement formel d'informations

Cette "machine" va donc s'atteler à des tâches de "traitement d'informations". Pour préciser cette assertion, voici quelques exemples de tâches, que nous connaissons bien, pour y être parfois confrontés, et qui mettent en oeuvre un traitement formel d'informations :

- additionner deux nombres;
- calculer une moyenne;
- chercher le plus grand d'une série de nombres;
- ...
- trier un paquet de cartes numérotées et qui se présentent dans le désordre;
- transcrire en toutes lettres un nombre écrit "en chiffres";
- compter et annoncer les points au tennis;
- chercher le numéro de téléphone d'un abonné;
- ...
- conjuguer un verbe régulier du premier groupe à tous les temps de l'indicatif;
- lire un texte et fournir la fréquence des divers mots le composant;
- centrer un titre lors de la dactylographie d'un document;
- ...

Le premier ensemble de tâches concerne des traitements de nombres; et les manipulations de données numériques sont, par essence, formelles. Le processus d'addition de deux nombres n'a que faire du "sens" qui serait attaché aux quantités à additionner : 13 + 12 cela fait toujours 25 quelle que soit la signification associée à ces nombres, qu'il s'agisse de poids, de sommes d'argent ou d'âges. L'ordinateur, manipulateur formel d'informations,[3] va donc exceller lorsqu'il s'agira de traiter des données numériques. Les opérations qu'il effectuera (ou plutôt qu'on lui fera effectuer) seront d'ailleurs similaires à celles que nous réalisons à propos des nombres : les additionner, les soustraire, les comparer, ...

> Je ne dis pas qu'avant et après le traitement formel de nombres nous n'attachons pas un sens aux données traitées mais la manipulation elle-même est indépendante de cette signification ! Il est exclu d'avoir des états d'âme lorsqu'on additionne des nombres !

Le deuxième ensemble de tâches correspond lui aussi, même si les informations traitées ne sont plus numériques, à des traitements formels : trier, chercher un numéro de téléphone, classer, ... sont des actions qui ne font appel qu'à la forme des données manipulées. ANTOINE précèdera toujours BENOIT dans un classement alphabétique et c'est l'apparence externe de ces prénoms qui permet d'en décider et non le fait que BENOIT est notre ami et ANTOINE un parfait inconnu. Évidemment, si dans cette liste de "prénoms" nous trouvons "AAFFRR", nous hésiterons à le classer avant ANTOINE, même si la forme nous y oblige et cela parce que le prénom "AAFFRR" nous paraît insensé; mais si nous excluons ces considérations de "sens", notre classement ne s'appuiera de fait que sur la forme des prénoms proposés.

3 L'ordinateur, même "multimédia", était, est et restera un **calculateur** électronique : il manipule des (représentations physiques de) **nombres**.

Le troisième ensemble de tâches nous est nettement moins familier. Il s'agit ici de manipulations formelles de texte. Et la plupart du temps, lorsque nous nous intéressons à du texte, ce n'est pas la forme qui nous importe mais le sens que nous lui attachons. Ainsi, nous ne "lisons" (généralement) pas "Germinal" ou "Notre-Dame de Paris" pour le plaisir de pouvoir déclarer que l'un comporte 1.248.368 mots et l'autre 1.168.413 ! Nous n'apprécions pas la portée d'un essai ou la beauté d'un poème sur base du nombre de fois que le mot "et" s'y trouve présent !

Lorsqu'il s'agira de "texte", l'ordinateur, manipulateur formel, ne pourra se livrer qu'à des opérations portant sur les caractères qui le constituent, sans référence, à la signification véhiculée. Nous parlerons d'ailleurs toujours de "chaînes (ou successions) de caractères" et non de "mots".

On voit que dans les tâches énumérées ci-dessus, j'ai soigneusement évité des travaux comme "résumer un texte", "traduire un texte", "apprécier la poésie se dégageant d'un texte". Ce ne sont plus là ce que j'appelle des traitements "formels" (= ne s'attachant qu'à la forme du texte et pas à ce que nous nommons son "sens").

Il ne viendrait à l'idée de personne, pour résumer un texte, de se contenter de le réécrire en passant un mot sur deux (ce qui constituerait un traitement purement "formel"); pour résumer un texte, nous devons avoir saisi son "sens", avoir identifié les "idées" importantes, les "thèses" avancées, ... Et tous ces concepts renvoient à autre chose qu'à l'aspect "formel" du texte.

De même, pour traduire, il ne suffit pas de consulter un lexique où chaque mot français aurait son équivalent anglais. Une telle attitude "formaliste", qui consiste à remplacer (bêtement) un mot par son correspondant, conduit à des absurdités comme "He the door" pour "Il la porte"!

Une dernière anecdote fera mieux comprendre ce caractère "formaliste" des traitements effectués par l'ordinateur. Il existe aujourd'hui, associés à certains logiciels de traitement de texte, des "correcteurs" orthographiques qui, pour chaque mot du texte, vérifient que ce mot est bien présent dans une (énorme) liste de mots (reprenant la plus grosse partie des mots acceptables). Dans le cas où le mot "fôte" est repéré dans le texte, l'ordinateur détecte bien une erreur... mais propose comme liste de mots possibles "forte", "fonte" ou "foetus"; et face à "saurtir" suggère "sautoir" ou "saurait" ! Par contre, rien dans la phrase "il faut pou voir se rencontrer" n'éveille une "réaction" de sa part, puisque le mot "pou" est dans la liste des mots acceptables, comme aussi le verbe "voir"...

Retenons donc que

L'ordinateur n'est capable que de traitements *formels* d'informations.[4]

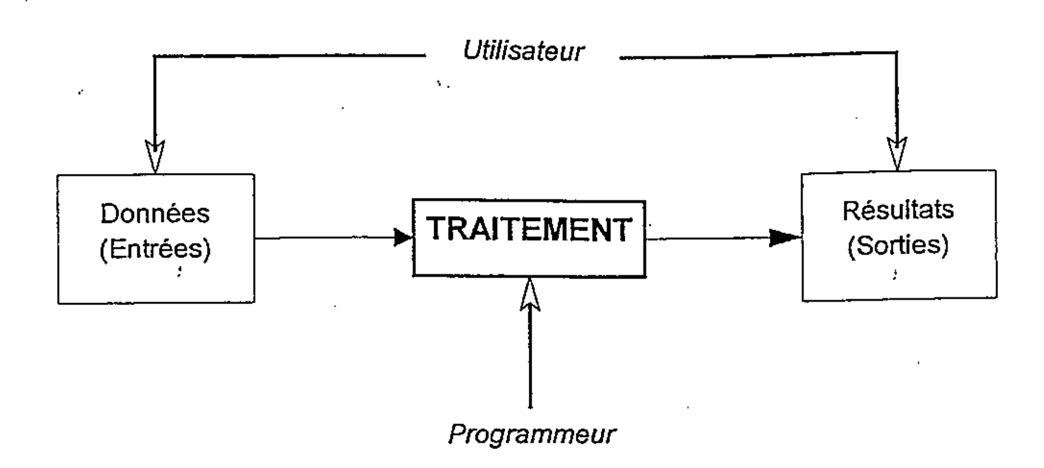
Nous pourrons déléguer à l'ordinateur l'exécution de ces tâches de traitement formel d'informations, à condition qu'il dispose de la "marche à suivre" qui le rende capable d'effectuer ces tâches. Et c'est là une facette essentielle : l'ordinateur sans programme (= marche à suivre explicative) pour le gouverner n'est capable de rien; c'est un principe fondamental :

TOUT ce que fait un ordinateur, il le fait gouverné par une "marche à suivre", un programme.

1.3 Les deux points de vue possibles

Schématiquement, on peut représenter les choses de la manière suivante :

4 Les lecteurs intéressés par les débats "philosophiques" sur "forme" et "sens", sur "information" et "connaissance", sur "syntaxe" et "sémantique", consulteront avec profit l'excellent livre de Jacques ARSAC, Les machines à penser (Cf. [7]).



L'utilisateur voit dans l'ordinateur, équipé des programmes convenables qui le feront agir, un outil : outil pour composer, modifier et présenter du texte; outil pour calculer des trajectoires balistiques; outil pour gérer les opérations d'emprunt de livres dans une bibliothèque; outil pour dessiner; ...

L'objectif, c'est alors d'apprendre à se servir au mieux de l'outil adapté à une tâche donnée; la demande exprimée par l'utilisateur potentiel, c'est "apprenez-moi le mode d'emploi de tel outil". Ce qui l'intéresse, c'est essentiellement : "quelles données dois-je fournir ?" et "quels résultats suis-je alors en droit d'attendre après le traitement qui y sera appliqué ?". Pour le reste, l'ordinateur équipé du programme (logiciel) qui le gouverne pendant l'utilisation, c'est une boîte noire.

Il est important de noter que ce que j'appelle "outil" c'est le tandem ordinateur-programme. Un ordinateur "nu" n'est capable de rien (Cf. le principe énoncé ci-dessus). L'outil, c'est l'ordinateur gouverné par le logiciel de traitement de texte ou l'ordinateur conduit par le logiciel de gestion de bibliothèque, l'ordinateur équipé du programme de calcul, etc..

En résumé,

pour l'utilisateur : ordinateur + programme } = outil

Le programmeur (ou plutôt l'analyste-programmeur) s'intéresse d'abord au traitement proprement dit. Pour lui, l'ordinateur est d'abord un exécutant à qui il va devoir fournir la marche à suivre (programme) "expliquant" le traitement à effectuer. L'accent est alors très nettement placé sur la tâche de traitement d'informations qu'on souhaite faire faire par l'ordinateur. La programmation, c'est essentiellement la création et la conception des outils (logiciels) qui seront utilisés par d'autres.

L'ordinateur cesse ici d'être une boîte noire ! Ce qui ne signifie nullement qu'on se passionne pour la "quincaillerie" (hardware) électronique qui le constitue : on y voit, non un ensemble de circuits intégrés ou de portes logiques, mais à travers le langage de programmation qui va le transfigurer (Cf. plus loin), un exécutant capable d'un certain nombre de traitements élémentaires et formels d'informations.

Ces deux perspectives débouchent évidemment sur des éclairages fort différents de l'ordinateur : outil (lorsqu'il est équipé du programme adéquat) pour l'utilisateur; exécutant, pour qui il faut décortiquer une tâche afin de lui indiquer le traitement à effectuer, pour le programmeur.

Autrement dit, on peut soit s'intéresser à ce dont l'ordinateur paraît capable (quand il est équipé du logiciel ad hoc), soit au problème de le rendre capable de ces activités en concevant pour lui les programmes lui "expliquant" le traitement à exécuter.

De ces deux approches possibles du monde des ordinateurs, c'est la seconde qui sera retenue et développée dans ces pages : qu'est-ce que "programmer", quelles sont les règles qui gouvernent la conception des "marches à suivre" qui nourriront l'ordinateur, comment "décortiquer" une tâche pour la faire exécuter ? ...

2. Programmer ?

2.1 Tâche ou problème?

Aux exemples de traitements (formels) d'informations signalés plus haut, on pourrait en ajouter bien d'autres comme :

- être capable de repérer dans une série de nombres quel est le plus petit et le plus grand;
- lire un texte et signaler ensuite combien de fois le mot "et" s'y trouve;
- jeter un dé jusqu'à ce que le 6 apparaisse pour la troisième fois et annoncer alors combien de fois il a fallu jeter le dé pour cela;
- écrire un nombre en chiffres romains;
- décomposer une somme d'argent en coupures (en minimisant le nombre de celles-ci);
- ... ...

Je pourrais, et de beaucoup, allonger cette liste. Mais, à chaque fois, les activités évoquées constituent des tâches, souvent fastidieuses, mais en tout cas d'une simplicité remarquable. Je n'ai jamais rencontré personne qui, face au travail consistant à trier 20 cartes fournies dans le désordre, ait demandé un long délai de réflexion ou s'en soit déclaré incapable. Que dire alors de l'activité consistant à additionner deux nombres par calcul écrit ou à compter les mots d'un texte !

Tous ces exemples décrivent des tâches, souvent élémentaires, parfois franchement risibles, tant elles paraissent dénuées de créativité et de la moindre réflexion. En tout cas, personne n'est prêt à qualifier de problèmes ces travaux rudimentaires. Le mot "problème" charrie avec lui une aura de difficulté, d'invention, de recherche que ne méritent absolument pas les tâches évoquées ci-dessus.

On ne "résout" pas le "problème" de lancer un dé jusqu'au moment où l'on obtiendra trois 6 de suite; on n' "analyse" pas longuement le "problème" consistant à repérer si un nom est oui ou non présent dans une liste triée alphabétiquement.

On m'objectera sans doute que s'il s'agissait de 20000 cartes à trier ou du décompte des mots constituant les oeuvres complètes d'Emile Zola, on se trouverait en face de vrais problèmes.

Qu'il soit malaisé de trouver quelqu'un prêt à s'atteler à des tâches d'aussi longue haleine et tellement fastidieuses, c'est vrai ! Mais ce n'est pas la complexité des tâches évoquées qui est en cause. Compter, par exemple, est une activité débile; compter beaucoup reste tout aussi idiot, mais s'avère en plus lassant, éreintant et accablant, ... mais cela ne devient pas pour autant un "problème". Si ces travaux étaient bien payés, nul doute que les candidats s'en déclarant capables ne manqueraient pas. Combien en resterait-il s'il s'agissait de fournir la solution à de petits problèmes d'arithmétique ?

Il est faux de dire qu'en programmation on "résout des *problèmes*" ou même que, pour programmer, il faut d'abord bien "analyser le *problème* posé". On s'intéresse le plus souvent à des *tâches* (souvent assez bêtes et fastidieuses).

On ne "résout" pas une tâche, on l'effectue !

Il y a 40 ans que l'informatique existe et, au risque de paraître provocateur, je dirais volontiers que

L'informatique n'a (presque) jamais "*résolu*" un seul "*problème*".

Je pratique l'informatique et la programmation depuis suffisamment longtemps pour mesurer (tout de même) ce que ces propos peuvent avoir d'outrancier et d'exagéré. Je peux évidemment apporter des exemples de tâches tellement longues et fastidieuses qu'il serait bien difficile de trouver un être humain acceptant de s'y atteler (tâches de calcul, de tri de grosses quantités d'informations, ...)(Cf. ci-dessus). Mais l'ordinateur ne résout pas ces tâches. Il les effectue vite et bien, c'est tout.

Par ailleurs, il est évident que, lorsque la tâche consiste à "gagner aux échecs" ou même simplement "gagner au tic tac toe", nous sommes prêts alors à parler de "problème" : la plupart d'entre nous sont d'ailleurs incapables de "résoudre ces problèmes" (= de gagner à coup sûr). Que dire évidemment dans ce cas du "problème" de "faire gagner l'ordinateur aux échecs" !!!!

Si j'insiste tellement ici sur cette distinction entre tâche et problème, c'est que j'ai encore en mémoire les yeux étonnés et le regard incrédule de mes premiers "élèves" et de mes collègues (non-informaticiens), à qui je parlais du "problème (sic) de conjuguer à l'indicatif présent un verbe régulier du premier groupe" ou du "problème, plus compliqué, (resic) de compter tous les mots comportant une lettre redoublée dans un texte".

2.2 Programmer, c'est faire faire ...

Et pourtant, chacune des tâches évoquées ci-dessus va tout de même donner lieu à un réel problème lorsque la programmation va s'en mêler.

En effet, il s'agira à chaque fois pour le "programmeur", non d'exécuter ces tâches lui-même (ce qui le plus souvent n'aurait aucun intérêt et serait affreusement fastidieux), mais de les faire exécuter. C'est cela, programmer ! Il s'agit, non d'être capable de mener à bien soi-même une tâche donnée, mais d'expliquer à "un autre" comment il doit s'y prendre pour exécuter cette tâche à notre place. En quelque sorte,

PROGRAMMER,
c'est
FAIRE FAIRE

Chacune des tâches décrites pose un véritable problème, dès qu'il s'agit de la faire faire par un autre. Évidemment, la difficulté dépend de manière cruciale des possibilités et des caractéristiques de cet "autre".

Première difficulté : l'exécutant a des capacités limitées;

S'il s'agissait d'un être humain, lui faire exécuter les diverses tâches évoquées, exigerait seulement qu'on lui dise

- repère dans la série de nombres qu'on va te fournir quel est le plus petit et le plus grand;

- lis le texte et signale ensuite combien de fois le mot "et" s'y trouve;
- jette un dé jusqu'à ce que le 6 apparaisse pour la troisième fois et annonce alors combien de fois tu as dû jeter le dé pour cela;
- écris en chiffres romains les nombres que je te fournirai;
- ...

Exactement comme il suffit de préciser "confectionne une blanquette de veau" pour obtenir le résultat souhaité, lorsqu'on s'adresse à un cuisinier confirmé.

Si, par contre, on est au prise avec un débutant (ou une débutante), cette petite phrase ne suffit plus : il est alors indispensable de fournir la recette correspondante. Et le détail des explications qui doivent être données dépend de manière cruciale des possibilités du cuisinier débutant.

L'ordinateur, s'il est bien un traiteur d'informations, peut malheureusement, quand on le regarde du point de vue de la programmation, être assimilé à un "débutant". Ses possibilités, on va le voir au chapitre 3, sont extraordinairement limitées. Et il est absolument hors de question qu'il "comprenne" des phrases comme "Trie les nombres qu'on va te fournir" ou "Conjugue au présent de l'indicatif un verbe qui sera précisé"..., même si on les lui dit en anglais ! S'il fallait dès à présent qualifier l'exécutant-ordinateur, les mots qui me viendraient tout naturellement en tête seraient plutôt "borné", "bête", "ne comprenant jamais rien à demi-mot", ...

> Je ne suis évidemment pas dupe de ces anthropomorphismes à propos de l'ordinateur. Si nous le découvrons "bête" ou "borné" quand nous nous intéressons à "lui" à travers la programmation, nous serions par ailleurs tenté de le qualifier d' "efficace" et parfois même d' "intelligent" quand nous posons sur lui un regard d'utilisateur. C'est que, dans ce dernier cas, il est évidemment transfiguré par les indications (le programme) qui lui précisent comment traiter aussi "intelligemment" les tâches considérées.
>
> L'une et l'autre de ces attitudes sont sans fondement réel : il n'y a guère plus de sens de parler d'un ordinateur "borné" qu'à prétendre que ma tondeuse à gazon est "découragée" et il est aussi inadéquat de le qualifier d' "intelligent" que de parler d'une lessiveuse "capricieuse" ou d'une bêche "courageuse"...

Le problème en programmation ce sera donc, face, d'une part à une tâche (même nous apparaissant comme simple), et face, d'autre part, à un exécutant aux capacités limitées (l'ordinateur), d'expliquer à ce dernier comment effectuer cette tâche à notre place.

Le problème ne réside pas (en général) dans la tâche qui sert de point de départ, mais dans le fait que nous devrons la faire effectuer par un exécutant qui, initialement, n'en est pas capable. La difficulté est donc de lui fournir les indications nécessaires pour que, s'y conformant, il paraisse capable de mener cette tâche à bien. Programmer est donc un verbe à rapprocher plutôt d'expliquer, de décortiquer, de déplier (déplier = ex-pliquer).

Deuxième difficulté : on "fait faire" en différé

Programmer, c'est "faire faire". Ce "faire faire" n'est cependant pas un "faire faire" en direct. On ne nous permettra malheureusement pas d'être aux côtés du cuisinier débutant, pendant sa prestation, pour lui fournir les indications nécessaires. Nous pourrions dans ce cas réagir à ses actions, ajuster nos explications, rectifier le déroulement de l'exécution,...

Ce qui sera permis, c'est seulement de fournir la recette à l'apprenti cuisinier; nous serons ensuite forcés de le laisser seul aux prises avec la confection de la blanquette de veau, sans aucune possibilité de modifier des instructions inadéquates ou insuffisantes, sans pouvoir intervenir dans l'exécution. TOUT doit être prévu dans la recette conçue, rien ne doit être laissé à l'appréciation de l'exécutant. Il n'est capable d'aucune initiative : tout au plus, dans le cas d'une recette incorrecte, son activité s'interrompra ... ou débouchera sur un plat immangeable et qui n'aura plus que de lointains rapports avec la blanquette de veau qu'on souhaitait initialement faire confectionner.

Lorsqu'il s'agit d'indiquer à quelqu'un comment il peut parvenir en voiture jusque chez moi, cela ne pose aucun problème lorsque je suis assis à côté du conducteur pour lui donner "en direct" toutes les instructions nécessaires. Mais, s'il s'agit de lui "expliquer" à l'avance comment se rendre seul chez moi, ceci nécessite alors beaucoup plus de soin et de rigueur : fournir la "marche à suivre" qui lui permettra d'arriver à coup sûr est bien compliqué, surtout s'il est borné au point de ne pas savoir lire une carte et m'oblige ainsi à transcrire par écrit toutes les indications indispensables...

Programmer, c'est donc rédiger une *marche à suivre* pour faire faire une tâche par un exécutant aux capacités limitées.

Je préfère le terme "marche à suivre" à celui plus classique d'"algorithme". D'abord, la coloration de ce dernier est fort "mathématique" : la plupart des gens pensent - à tort - que le mot "algorithme" entretient d'étroits rapports avec "logarithme". Ensuite, il laisse davantage dans l'ombre, me semble-t-il, la nécessité d'avoir précisé l'exécutant.

On pourrait aussi parler d'ensemble de "consignes", de série d'indications ou d'instructions. De toute manière, ce concept de marche à suivre sera bientôt affiné et plus complètement abordé puisque le chapitre 2 tout entier y est consacré.

Le problème, c'est donc d'être capable de morceler une tâche "complexe" (si on la compare aux possibilités de l'exécutant-ordinateur) en une succession de tâches plus élémentaires (= dont l'exécutant est capable). Il faudra organiser la succession de ces dernières pour que, les effectuant dans l'ordre indiqué, l'exécutant ait finalement accompli la tâche complexe initialement proposée.

Il nous arrive parfois d'exécuter des marches à suivre lorsque, par exemple, nous suivons les indications d'une recette de cuisine ou d'un guide de tricot. C'est aussi le cas de l'exécutant-musicien face à une partition-marche à suivre. Chacun mesure cependant, dans ce cas, toute la distance entre ce statut d'exécutant et celui du compositeur-programmeur, dont le rôle est de concevoir et d'écrire les partitions-marches à suivre. Ceci pour insister sur le fait que cette activité (rédiger une marche à suivre) est totalement nouvelle et inconnue pour la plupart d'entre nous (sauf pour les auteurs de guides de tricot ou de livres de recettes !).

Troisième difficulté : l'exécutant est un "robot";

Tous les exemples mentionnés ci-dessus sont en partie trompeurs : c'est qu'il s'agit à chaque fois de fournir les instructions nécessaires à un être humain, qu'il soit apprenti cuisinier, musicien débutant ou tricoteur (tricoteuse ?) novice. Même si la marche à suivre à concevoir doit être complète (en envisageant toutes les situations possibles) et non ambiguë (en ne laissant aucune initiative à l'exécutant), notre interlocuteur est un être humain, comme vous et moi.

Dès lors, nous pourrons, dans certaines limites, nous exprimer à demi-mots, garder certains implicites (partagés par tous les humains), adopter une représentation imagée ou dessinée, ... Tout ceci sera évidemment exclu lorsque l'exécutant des marches à suivre à concevoir sera un "robot". Dans ce cas, les exigences de précision, de chasse aux implicites, d'exhaustivité seront pratiquement "inhumaines".

Il est un point cependant où le caractère "borné" de l'exécutant-robot est intéressant et même indispensable. On comprend aisément que pour qu'un problème de programmation soit correctement posé, il faut disposer non seulement de la description précise de la tâche à faire effectuer mais aussi de celle de l'exécutant à qui on la destine. C'est dire qu'il faut connaître complètement et précisément tout ce dont ce dernier est capable : c'est possible pour un robot aux actions limitées, mais non pour un être humain ! Ainsi, concevoir une marche à suivre destinée à un humain est un exercice périlleux, puisque, ne disposant pas d'un portrait complet de ses possibilités, nous ne saurons jamais avec certitude, a priori, si nos consignes sont insuffisantes ou inutilement détaillées.

Je voudrais revenir et insister sur le fait que pour qu'un problème soit bien posé en programmation, il faut non seulement que la tâche qu'il va falloir faire faire soit complètement précisée, mais encore que l'exécutant à qui je devrai fournir les consignes explicatives soit parfaitement défini.

On propose pourtant fréquemment aux débutants des énoncés du type

"analyser le problème de la mise en marche d'une voiture"

ou

"écrire l'algorithme pour la préparation du café"

ou

"résoudre le problème consistant à traverser une rue sans se faire écraser par les voitures".

On y désigne comme des "problèmes" des activités bien familières. On me rétorquera qu'il s'agit de décrire la manière dont on s'y prend pour effectuer ces divers travaux. Jusqu'où alors pousser les explications et le décorticage ? En réalité, ce qui est souhaité, c'est qu'on décrive les algorithmes (ou marches à suivre) qui sous-tendent chacune de ces tâches. Mais, dans ce cas, les énoncés proposés sont notoirement incomplets puisqu'on n'a pas précisé, à chaque fois, les capacités de l'exécutant à qui on s'adressera. Que comprend et que peut faire l'apprenti conducteur ou l'exécutant-piéton ? Sans précision supplémentaire, il est impossible de savoir jusqu'où ces diverses tâches doivent être disséquées.

On devine déjà qu'avant d'aborder l'analyse du moindre problème de programmation, il faudra avoir dit quelles sont les caractéristiques de l'exécutant-ordinateur, sinon la démarche toute entière est viciée : il est impossible de faire faire quelque chose par un autre, sans savoir ce dont il est capable !

3. Les étapes du faire faire

Nous savons à présent que l'essentiel de ces notes a pour but de nous apprendre comment faire faire une tâche (de traitement formel d'informations) par "un autre". La tâche à faire réaliser est généralement anodine. Le problème vient de ce que celui à qui nous souhaitons faire exécuter ce travail n'en est pas directement capable. Il nous faut décortiquer, morceler la besogne en une succession d'actions plus élémentaires correspondant aux capacités de l'exécutant. Ce décorticage, ces explications s'incarnent dans une marche à suivre qui guidera son activité lorsque nous exigerons qu'il effectue cette tâche.

Schématiquement, le cœur du processus se décrit comme :

Tâche ⟶ **Faire faire** ⟶ **Ordinateur**

Sur le chemin qui conduira de la tâche à son exécution par l'ordinateur, plusieurs étapes sont indispensables. Et d'abord, il faudra répondre à la question suivante :

3.1 Quoi faire ?

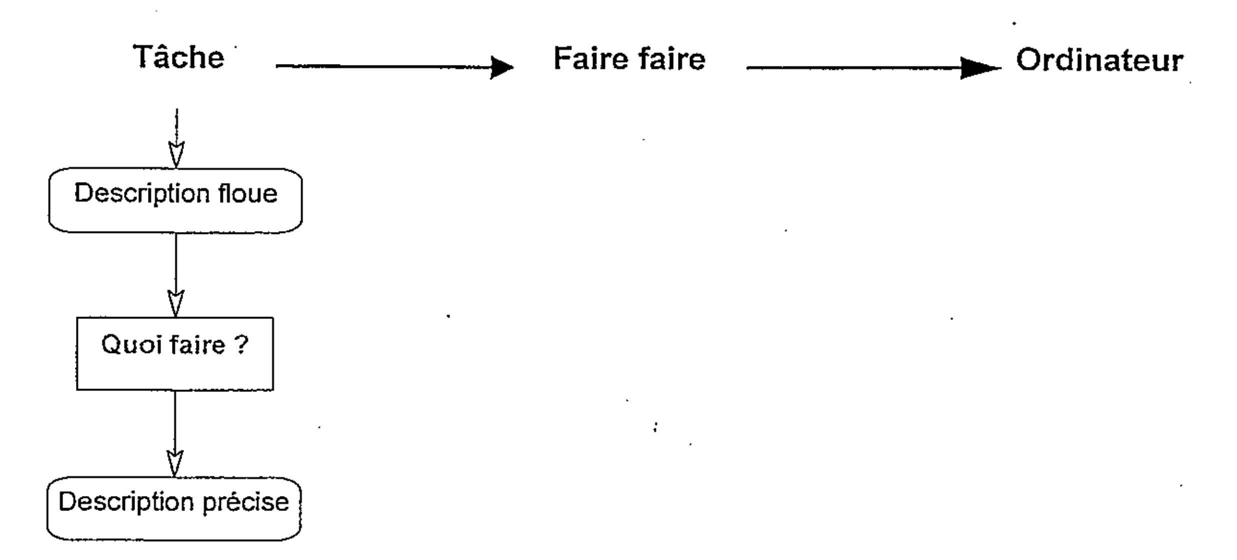
En effet, la tâche se présentera généralement d'abord de manière très floue. Il sera donc primordial, dans un premier temps, d'affiner sa description. Ainsi, face à la besogne consistant à écrire un nombre en toutes lettres, une série de questions viendront préciser ce qui doit être fait :

- De quelle taille seront les nombres à écrire ? Plus petits que cent ? Inférieurs à un million ? A un milliard ?
- Les nombres comporteront-ils une partie décimale ? Dans ce cas, comment exprimer cette dernière ?
- Faut-il les écrire "à la belge" (septante-huit) ou "à la française" (soixante-dix-huit) ?
- Faut-il les écrire en majuscules ? En minuscules ?
- ...

On le voit, même pour une tâche simple, il est indispensable de préciser ce qui est attendu, ce à quoi on s'engage.

Il est complètement illusoire de croire qu'on va pouvoir faire faire quelque chose par un autre si l'on ne sait pas précisément ce qu'il faut faire (faire) !!!!!

Cette étape du "Quoi faire ?" est en quelque sorte celle du "cahier des charges", celle de la recherche des "spécifications" (pour parler comme les informaticiens). C'est celle qui nous fera passer de la description floue de la tâche à sa description précise.



1. On passe très souvent sous silence cette première étape dans les initiations à la programmation. C'est fort dommage et cela donne, en tout cas, une idée complètement tronquée de ce que recouvre la démarche informatique d'analyse (et de programmation). Les

tâches qui seront abordées ici resteront évidemment de petite taille et d'une complexité fort raisonnable. Je ne fournirai pas de grands principes méthodologiques permettant de guider la démarche sous-jacente à cette étape : un jeu de questions et réponses nous tiendra lieu de méthode. Comme il s'agira à chaque fois de tâches consistant à traiter des informations, nous orienterons les questions dans trois directions :

- les données nécessaires au traitement envisagé (entrées) (leur forme, leur nombre, ...);
- les résultats souhaités (sorties) (forme, présentation, ...);
- le traitement (en quoi les résultats sont liés aux données).

Mais, on le verra, cette étape de précision de ce qui est attendu sera toujours la première et nous n'en ferons jamais l'économie.

2. Lorsque quelque chose "ne tourne pas rond" dans l'utilisation de l'ordinateur, par exemple dans la gestion d'une entreprise, ce n'est (presque) jamais à cause de l'ordinateur lui-même (rien n'est moins "capricieux" qu'un ordinateur). C'est parfois parce que les programmes sont incorrects (ils ne font pas faire par l'ordinateur ce qu'on souhaite ou tentent de lui faire faire des choses impossibles). Mais, le plus souvent, c'est parce qu'on n'a pas consacré le temps nécessaire à préciser ce qui était attendu. On se retrouve en bout de course avec un produit qui ne donne pas satisfaction essentiellement parce que, tout bêtement, "on" n'a pas dit TOUT ce que l'on voulait.

3. Dans la programmation telle qu'elle est vécue par les amateurs (ou les apprenants) la tâche abordée et la connaissance des possibilités offertes (par l'exécutant) pour la traiter sont "dans la même tête". Les questions sont posées par celui qui connaît aussi les réponses. Dans la réalité professionnelle, l'informaticien (analyste) débarque généralement dans un milieu qu'il ne connaît pas et qu'il doit "informatiser". Il n'y a pas grand chose de commun entre la tâche consistant à gérer le stock d'un magasin de chaussures et celle de la gestion des emprunts dans une bibliothèque ou de l'organisation d'un cabinet de dentiste ! Et celui qui "sait" ce que recouvre précisément chacune de ces tâches, c'est le client qui ne sait pas qu'il sait : il est généralement incapable d'expliciter comment il s'y prend actuellement et encore moins de décrire ce qu'il souhaite (il "fait" mais ne sait pas "comment il fait"). Et en face de lui, il y a l'informaticien qui, lui, connaît les possibilités de l'ordinateur, mais ne sait pas ce qu'il doit en faire dans ce cas précis. En résumé, il y a dans une tête (celle du client) les réponses, mais il ne sait pas qu'elles y sont; et dans une autre tête (celle de l'informaticien) il y a des questions, mais il ne sait pas bien celles qu'il est pertinent de poser.

   Le bon informaticien n'est pas celui qui commence par apporter des réponses mais celui qui pose d'abord les bonnes questions !

4. Ce n'est pas un hasard si la formation des informaticiens comporte une grosse partie consacrée à l'acquisition de méthodologies d'exploration du "Quoi faire ?" pour de grandes catégories de tâches : gestion, conception de bases de données, ... Il s'agit là de la première étape, souvent la plus ardue, du travail de l'informaticien, sur le chemin qui conduit de la tâche à son traitement par l'ordinateur.

5. Les questions posées viendront de deux horizons :
   - Certaines seront motivées par la tâche elle-même dont il faut affiner la description, indépendamment du fait qu'il faille ou non la faire traiter par un ordinateur. Ce sont les questions que nous poserions de toute manière si l'on nous demandait d'effectuer la tâche nous-mêmes.

- D'autres précisions sont exigées parce qu'il s'agira de faire faire les choses par l'ordinateur. Ainsi, dans l'exemple de l'écriture des nombres en toutes lettres, il est impératif de savoir, dans le cas où les nombres à écrire comportent une partie décimale, si cette dernière sera annoncée par une virgule (comme en français) ou par un point (comme les anglo-saxons). Ce qui n'est qu'un détail futile, auquel nous ne penserions même pas s'il s'agissait d'écrire les nombres nous-mêmes, devient important parce qu'il va s'agir de les faire écrire (l'exécutant-ordinateur préférant souvent alors le point à la virgule). Cette deuxième catégorie de questions est bien entendu la plus difficile à découvrir pour le débutant puisqu'elle postule qu'on connaisse bien les caractéristiques de l'exécutant-ordinateur qui va les motiver.

A l'issue de cette première étape, le cahier des charges est établi : nous savons alors, avec précision, ce qu'il faut faire (faire). L'étape suivante est évidemment celle du

3.2 Comment faire ?

Il s'agit ici de mettre au jour les stratégies employées : comment, face à la tâche précisée à l'étape précédente, nous y prenons-nous ? Il y a malheureusement un monde entre "être capable de..." et "pouvoir expliquer comment on s'y prend pour ...".

> "Solving a problem gives you the solution. Knowing how you solved the problem gives you a program."
>
> M. JAMES dans [43]

Cette petite phrase illustre bien toute la distance existant entre la capacité d'effectuer une tâche et la conscience qu'on a des stratégies mises en oeuvre pour y parvenir. Si je vous demande d'expliquer comment vous vous y prenez pour trier un (petit) paquet de cartes dans le désordre, il est inutile que vous recommenciez à nouveau l'opération sous prétexte que j'ai mal vu. Je sais fort bien que vous pouvez trier ce paquet de cartes et il ne sert à rien de le faire et le refaire : ce que j'attends c'est que vous expliquiez complètement et dans le détail comment vous procédez, que vous mettiez au jour les stratégies employées. C'est ici qu'il faut être capable de "faire le petit pas en arrière" qui permet de se "regarder faire"; c'est ici qu'il faut, en quelque sorte, quitter son rôle d'"acteur" pour adopter celui de "metteur en scène".

1. Même pour des tâches anodines (comme procéder à l'addition écrite de deux nombres ou compter les points au tennis) il n'est pas simple d'indiquer les stratégies employées. Et les tâches pour lesquelles nous avons le plus de mal à mettre au jour ces stratégies sont probablement celles que nous effectuons le plus automatiquement ou le plus inconsciemment : nous avons à ce point intégré les comportements sous-jacents que la question "comment fais-tu ?" nous prend au dépourvu et nous paraît en tout cas bien incongrue.

2. Il est des tâches un peu plus compliquées, comme écrire un nombre en toutes lettres ou décider si un nombre est premier, qui nous poseront probablement déjà de petits problèmes même s'il s'agit seulement de les effectuer nous-mêmes sans être tenus d'expliquer en détail comment nous nous y prenons. Dans ce cas cependant, les livres que nous consulterons ne sont absolument pas les traités d'informatique. C'est dans une grammaire que nous trouverons les règles permettant d'écrire sans erreur un nombre en toutes lettres et dans un livre d'arithmétique que nous apprendrons (ou réapprendrons) ce qu'est un nombre premier et comment on peut vérifier cette caractéristique. Ainsi, face à une tâche qui nous pose un

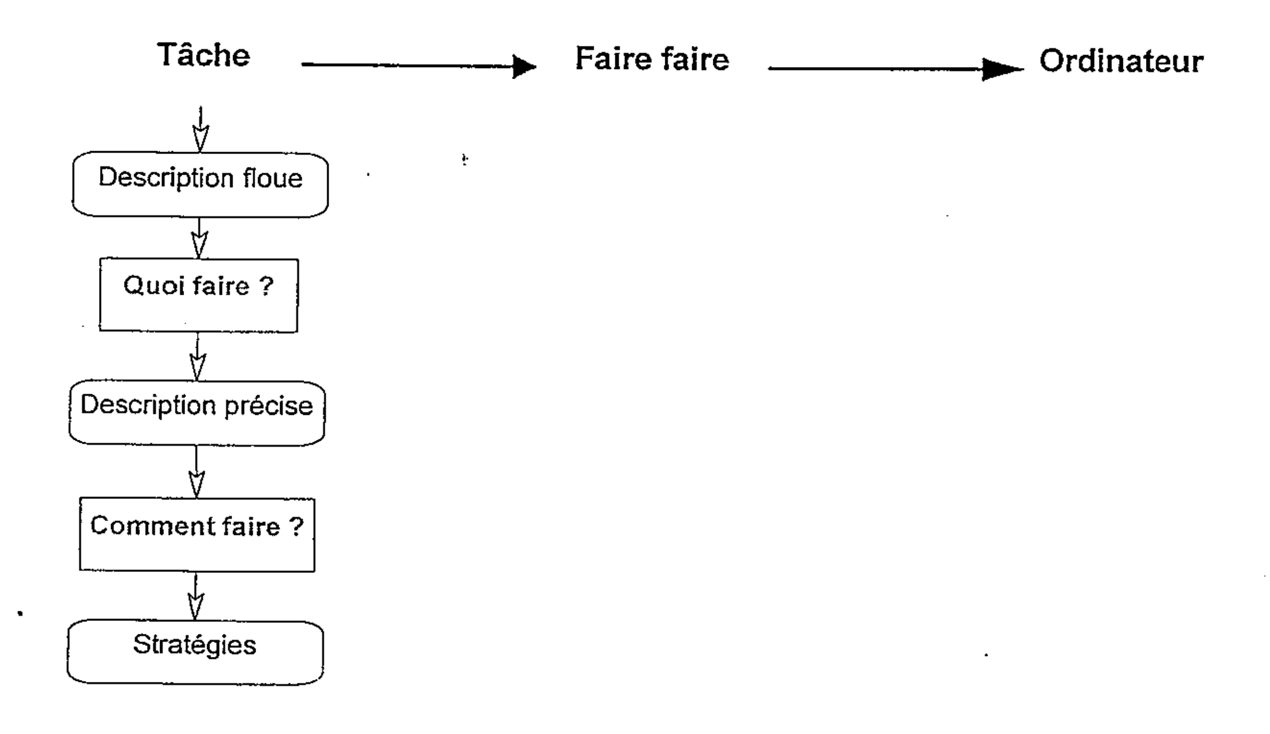
problème, les stratégies sont à chercher dans le champ (domaine) de cette tâche, pas dans les traités d'informatique !

3. L'attitude clé ici, c'est d'être capable de passer de l'implicite à l'explicite.

   C'est un fait que, trop souvent, nous avons appris à effectuer des tâches ou à résoudre des problèmes mécaniquement, sans avoir mis au jour, explicitement, comment nous nous y prenons, sans les avoir complètement dépliés (déplié = ex-pliqué). La programmation oblige à cette prise de distance où l'on se voit aux prises avec la tâche et où l'on prend conscience de sa stratégie. Pour "faire faire", il faut d'abord mettre à plat son savoir faire.

   C'est un apprentissage extrêmement exigeant et difficile que celui de cette chasse sans pitié aux implicites, aux flous, aux "à peu près"...

   Le chemin qui conduit de la tâche à son exécution par l'ordinateur s'enrichit donc d'une étape supplémentaire :



3.3 Comment faire faire ?

Voici l'étape cruciale et absolument neuve : c'est celle où, sur base de la description précise de la tâche et s'appuyant sur les stratégies de traitement mises au jour, nous allons rédiger la marche à suivre destinée à l'exécutant ordinateur.

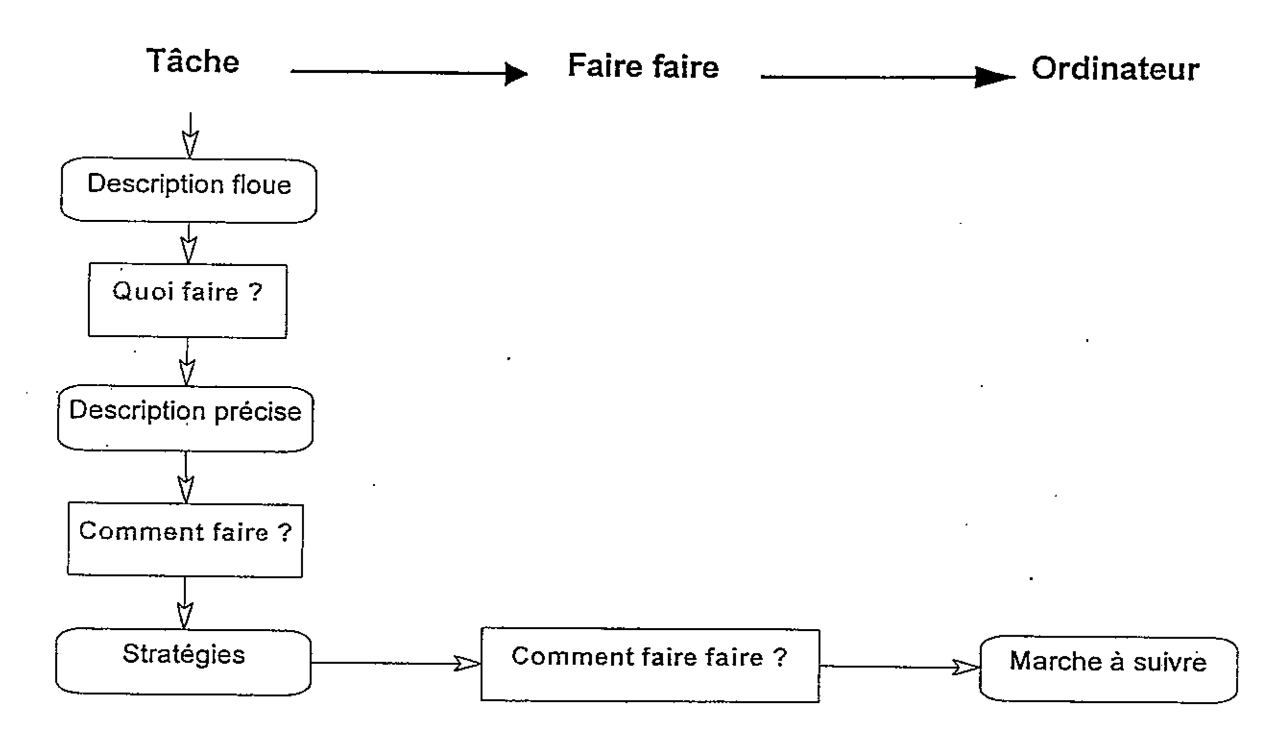
1. C'est bien entendu ici que la connaissance des capacités de cet exécutant devient indispensable : c'est seulement en sachant ce qu'il peut "comprendre" et faire que la rédaction d'une marche à suivre prend un sens. On verra que si le chapitre 2 précise les règles de conception et d'écriture de toute marche à suivre, le chapitre 3 répond à la question "Qui est l'exécutant-ordinateur ?". C'est seulement après avoir tracé son portrait que nous pourrons rédiger les premières marches à suivre qui vont le gouverner.

2. C'est évidemment toujours "en français" qu'à ce stade les marches à suivre seront conçues et exprimées. Bien entendu, on le verra, elles n'utiliseront pas toute la richesse d'expression permise par notre langue : nous y mettrons en évidence certains mots et en éviterons

d'autres; nous nous aiderons de graphismes bien choisis pour en montrer la structure;... Mais toujours, nous veillerons à déboucher sur une expression aussi compréhensible et claire que possible pour l'être humain qui serait amené à en prendre connaissance. Ainsi donc, nous serons attentifs, non seulement à nous soumettre aux capacités de l'exécutant à commander, mais aussi à tenir compte de ceux qui voudront relire et comprendre la marche à suivre rédigée.

3. C'est au cours de cette étape que nous nous aiderons d'une méthode particulière : l'approche descendante et structurée. Il est trop tôt pour détailler le contenu de cette manière de procéder[5]; j'ajouterai simplement que l'informatique a redécouvert là une méthode aussi vieille que l'humanité elle-même : face à un problème, on découpe celui-ci en sous-problèmes, plus aisés à traiter, puis ces derniers sont à leur tour morcelés, et ainsi de suite,... Cette attitude qui partira de la tâche dans son intégralité et sa complexité, pour descendre pas à pas, par affinements successifs, vers les actions élémentaires que nous pouvons exiger de l'exécutant, s'incarnera et se formalisera ici tout au long de l'apprentissage proposé.

   Le schéma se complète donc comme suit :



3.4 Comment dire ?

Nous disposons à présent, sous l'une ou l'autre forme (Cf. Chapitre 2), de la marche à suivre suffisante pour que, s'y conformant, l'exécutant-ordinateur effectue la tâche décrite.

Malheureusement, l'attention que nous aurons portée à rendre cette marche à suivre aisément lisible par d'autres, aura généralement conduit à l'écrire (ou même à la dessiner) sous une forme qui ne convient pas à l'ordinateur. Il va à présent falloir l'exprimer dans un langage particulier, "compréhensible" par l'ordinateur, dans un langage de programmation .

---

5 On pourra consulter le chapitre 2 ("Diviser pour régner") du second tome de "Images pour programmer".

1. Ces langages de programmation sont nombreux : vous connaissez sans doute Basic ou Pascal, Logo, Fortran, Cobol, ... Il en existe des dizaines, sinon des centaines ! Dans chaque cas, il s'agit bien d'un langage, construit de toutes pièces, et non d'une langue, comme celles à travers lesquelles les humains communiquent entre eux. Le vocabulaire de ces langages est (très) pauvre; les règles de grammaire qui les gouvernent sont arbitraires et (heureusement) peu nombreuses. Mais il nous faudra impérativement les respecter si nous souhaitons être "compris" par l'exécutant-ordinateur.

2. Inutile de souligner que ces divers langages ne sont pas les nôtres ! Aucun n'est d'ailleurs non plus vraiment celui de l'ordinateur ! On peut donc parler de langage compromis, puisque, ayant choisi un langage de programmation, je ferai l'effort d'y exprimer mes marches à suivre et que, par ailleurs, l'ordinateur acceptera que je m'adresse à lui dans ce langage qui n'est pas vraiment le sien.

   C'est dire qu'en tout cas, du point de vue de l'ordinateur, une traduction restera indispensable. Mais, heureusement, c'est lui qui s'en chargera. Ainsi, chacun "y met du sien" : j'accepte d'exprimer mes marches à suivre dans un langage qui n'est pas le mien et l'ordinateur, dont ce n'est pas non plus le langage "naturel", accepte de se charger de la traduction, pour déboucher sur une version qui, enfin, sera "la sienne" et qu'il pourra exécuter.

   Mon objectif n'est évidemment pas ici d'entrer dans des détails "techniques" à propos de ces problèmes. Il faut cependant savoir que le seul langage compréhensible par l'ordinateur (ou plus précisément par le processeur qui en est le "cœur" ou le "cerveau"[6] est le langage machine et non l'un quelconque des langages de programmation cités ci-dessus; c'est pour cela qu'une traduction est indispensable. J'ajouterai simplement que, comme toujours, si l'ordinateur paraît capable de traduire un texte qui lui est fourni dans l'un ou l'autre langage de programmation, c'est qu'il est alors gouverné par un programme qui lui "explique" comment effectuer cette traduction.

   Ainsi, dès que je dispose, pour un ordinateur donné, du programme qui va lui faire faire la traduction de tel langage vers le sien, je peux m'exprimer dans ce langage. L'impression que donne parfois l'ordinateur d'être "polyglotte", puisque la même machine va accepter des programmes rédigés en Basic, en Pascal, en Fortran, ... cache en réalité l'existence, pour chacun de ces langages, du programme de traduction correspondant.

   Ces programmes particuliers, capables de faire effectuer par l'ordinateur ces traductions s'appellent des compilateurs ou des interpréteurs.

3. Pourquoi ne pas avoir directement exprimé la marche à suivre dans ce langage compromis ? Les deux étapes du "Comment faire faire ?" et du "Comment dire ?" se trouveraient alors confondues. L'avantage paraît évident : nous nous exprimons d'emblée sous une forme qui peut être comprise par l'exécutant-ordinateur.

   Trois arguments essentiels s'opposent à cette vision simpliste :

   - La rédaction d'une marche à suivre dans l'un ou l'autre langage de programmation, si elle convient parfaitement à l'ordinateur, est assez éloignée de notre manière de nous exprimer. N'oublions pas que ces marches à suivre, si elles sont bien destinées à commander un ordinateur, devront aussi pouvoir être relues par un être humain (y compris par celui qui les a conçues). Il est donc préférable que la forme adoptée permette une lecture et une compréhension les plus aisées possible. Ce sera le cas pour la présentation adoptée à l'issue de l'étape du "Comment faire faire ?"; ce le sera

6 Rappelez-vous : la bêche courageuse...

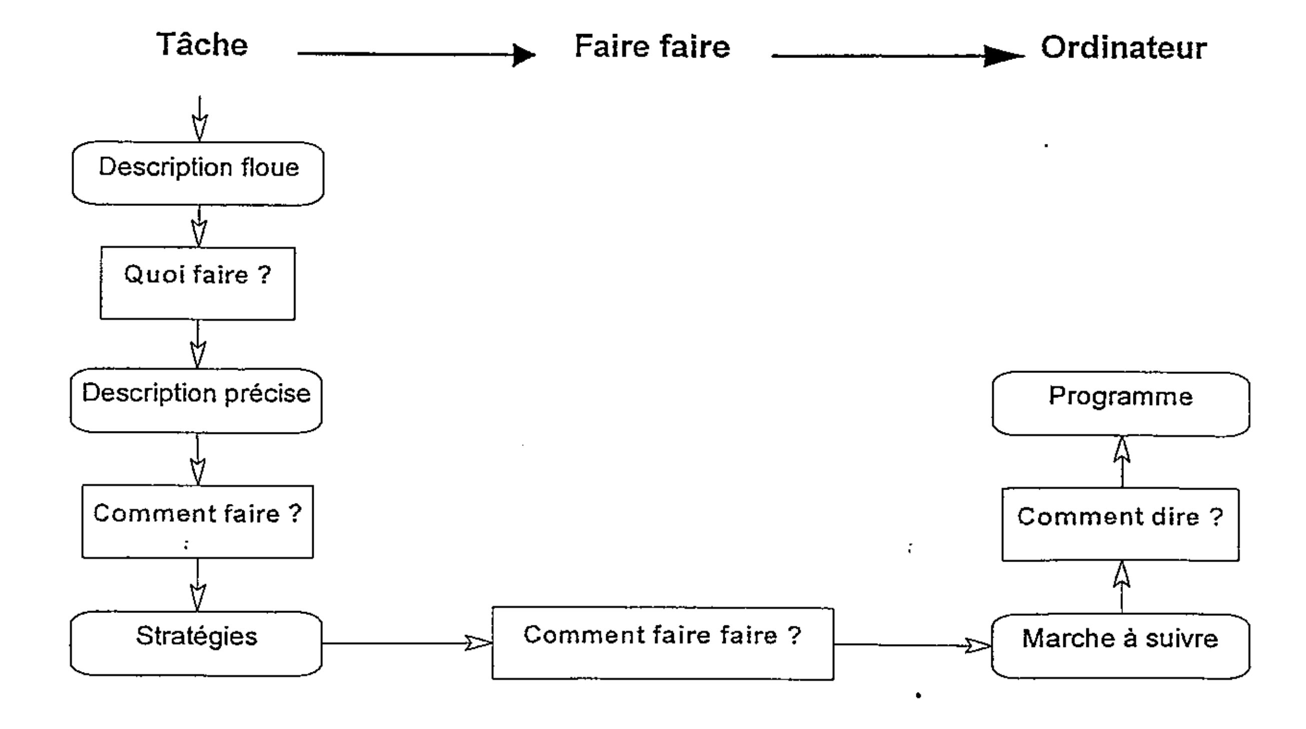
beaucoup moins lorsque la marche à suivre aura pris la forme d'un programme obligé de respecter toutes les règles syntaxiques et orthographiques imposées par le langage de programmation.

- Il est préférable de mêler le moins possible les difficultés de conception de la marche à suivre (qui sont, on le verra, considérables) avec celles inhérentes au respect de la syntaxe du langage de programmation. Il est, comme toujours, préférable de morceler les difficultés que de les aborder ensemble.
- Surtout, le mode d'expression retenu pour les marches à suivre, très largement indépendant du langage de programmation choisi, pourra donc être adapté à plusieurs de ces langages. En effet, le portrait que je tracerai de l'exécutant-ordinateur restera (pratiquement) identique quel que soit le langage dans lequel je finirai par devoir m'adresser à lui. Ainsi, la marche à suivre explicitant telle tâche sera la même que je finisse par m'adresser à l'ordinateur en Pascal ou en Basic, par exemple.

4. Les qualités dont il faudra faire preuve à cette étape sont bien différentes de celles nécessaires aux étapes précédentes. Ici, c'est le règne des conventions, de la grammaire et de l'orthographe (spécifiques au langage de programmation choisi); qualités de rigueur et de docilité, dès lors, et surtout pas d'invention, de créativité ou d'imagination ! Ce qui est indispensable, c'est de savoir quand la virgule ou le point sont requis; comment il faut orthographier les (quelques) mots du langage choisi, quelles sont les "tournures" permises ?

5. A l'issue de cette étape, au cours de laquelle, je le souligne, l'ordinateur n'est toujours pas indispensable, je dispose donc d'un programme. Un programme, c'est donc seulement une marche à suivre écrite dans un langage particulier. L'avantage de cette forme d'expression de nos marches à suivre, c'est qu'elle est "compréhensible" par l'ordinateur; son désavantage, c'est d'être nettement moins lisible pour nous !

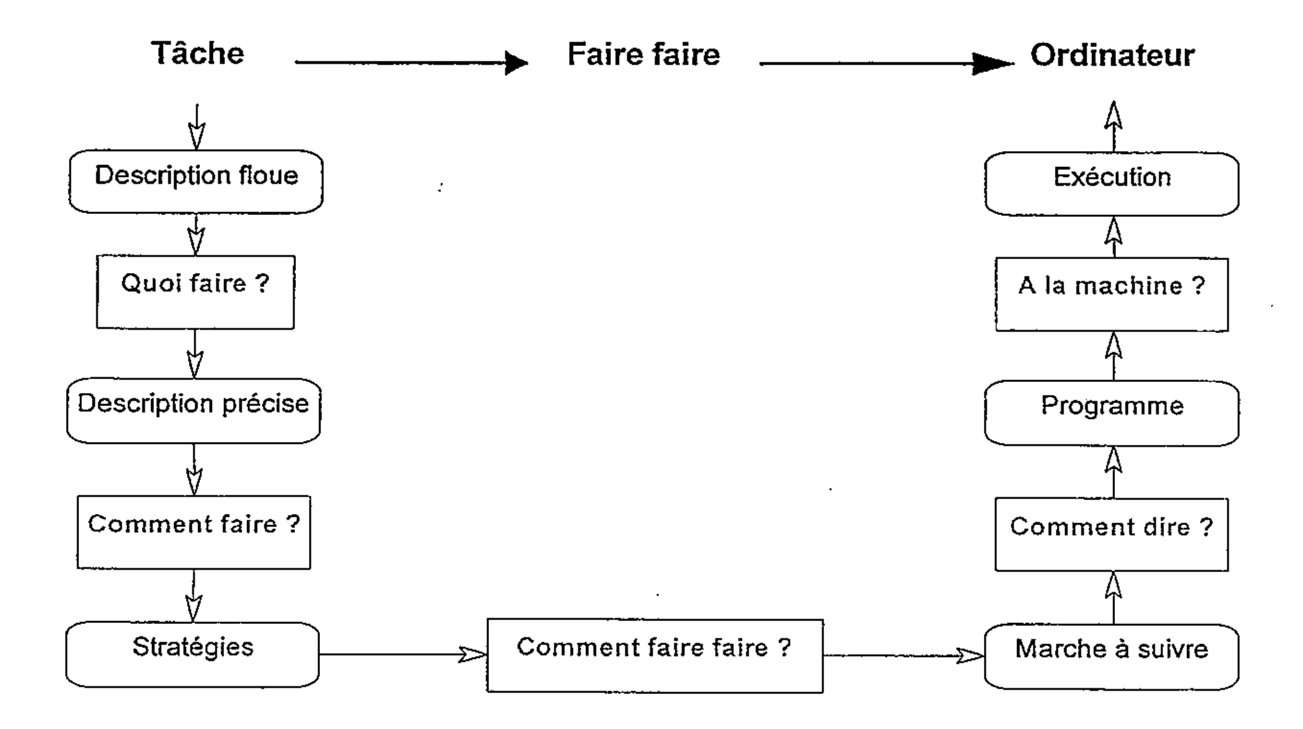
La description schématique s'enrichit donc d'une étape supplémentaire :



Jusqu'ici le travail d'analyse et de programmation ne nécessitait pas la présence effective de l'ordinateur. Disposant maintenant du programme conçu grâce aux étapes précédentes, je vais enfin me retrouver face à l'ordinateur pour lui fournir le résultat de mes cogitations et lui demander d'exécuter la tâche envisagée.

3.5 *Et face à la machine ?*

Nous voici donc au terme du parcours, aux prises avec la dernière étape, celle du travail effectif à l'ordinateur.



Ce travail comportera en général plusieurs phases :

1. Je vais d'abord fournir à l'ordinateur le texte de mon programme (exprimé dans le langage de programmation retenu). L'ordinateur se contente d'entasser ce texte dans sa mémoire, tout en m'aidant à le confectionner. Il est alors gouverné par un programme spécifique qu'on appelle un éditeur de texte. Ce programme le transforme en quelque sorte en une "super machine à écrire", à ceci près que le texte s'inscrit à l'écran et non sur une feuille de papier et qu'il prend simultanément place dans la mémoire centrale.
2. A l'issue de cette étape d'édition, le texte résidant en mémoire est généralement sauvé sur un support mémoire externe : cassette, disquette ou disque dur. Il y acquiert une existence permanente, contrairement à la version présente en mémoire vive qui peut disparaître, par exemple lors d'une coupure de courant, réduisant alors à néant un travail de dactylographie représentant quelques minutes... ou quelques heures. Le texte ainsi recopié sur support externe pourra bien entendu être ramené (on dit "chargé") dans la mémoire pour être réutilisé.
3. Nous savons que le texte du programme est, tel quel, incompréhensible, donc inexécutable, par l'ordinateur : il doit d'abord être traduit dans le langage de ce dernier (ce qu'on appelle le

langage "machine"). Pour effectuer cette traduction, l'ordinateur se laisse gouverner par un programme de traduction qu'on appelle programme **compilateur**; .

C'est au cours de cette étape de traduction ou de **compilation** que les erreurs de syntaxe présentes dans le texte sont détectées. Cette détection des erreurs est d'ailleurs approximative : l'ordinateur "se plante" pendant la compilation (il ne "comprend" plus un texte où l'on n'a pas scrupuleusement respecté les règles de syntaxe du langage compromis) et il signale l'endroit du texte où il a commencé à ne plus pouvoir traduire. Souvent, il assortit ceci d'une tentative de repérage du type d'erreur possible sous forme d'un message plus ou moins sibyllin ! Il reste alors à l'apprenti-programmeur à réexaminer le texte de son programme, par exemple en recommençant une phase d'édition de ce texte pour le modifier en y corrigeant la ou les erreurs ayant provoqué la fin prématurée de la traduction.

Ce cycle compilation-édition se poursuit tout le temps que des erreurs subsistent. Il peut être fort long si le programmeur est peu soigneux, distrait ou irrespectueux des règles d'écriture prescrites par le langage de programmation adopté. A l'issue de ces péripéties, on finit en général par terminer victorieusement l'étape de traduction et on se trouve alors avec en mémoire la version traduite du texte : on appelle parfois version "objet" ou version "code" le résultat de la compilation réussie pour la distinguer de la version texte ou version "sujet", celle exprimée dans le langage de programmation.

Il reste évidemment, si on le souhaite, à sauver sur support mémoire externe cette version directement exécutable de la marche à suivre.

4. Disposant maintenant de cette version, l'étape suivante consiste évidemment à demander (enfin !) l'exécution de la marche à suivre : l'ordinateur se laisse alors gouverner par la version traduite de cette dernière et effectue les traitements souhaités... du moins quand tout se passe bien ! C'est en effet au cours de cette étape qu'on découvre les erreurs de conception, les incohérences, bref ce qu'on désigne souvent sous le vocable d'erreurs de "logique" (Cf. plus loin).

Cette dernière étape achève la longue marche qui, chaque fois, conduira de la tâche à son exécution par l'ordinateur. Ce n'est ni simple, ni rapide... mais c'est cela programmer !

4. Quelques commentaires

4.1 Éviter la confusion des genres !

Je suis de cette génération de ceux qui "font de l'informatique" sans être informaticien (= sans avoir reçu une formation d'informaticien). Mon seul contact avec le monde de l'informatique au cours de mes études (de mathématique) a été une visite du centre de calcul de l'université : j'en ai seulement retenu le bruit et la rapidité des énormes imprimantes. Plus tard, quand j'ai eu besoin d'utiliser l'ordinateur (essentiellement pour calculer), on m'a dit "mais c'est bien simple, il suffit de trois jours de FORTRAN !", ce que j'ai cru ! Et pendant 10 ans, j'ai pensé que programmer, c'était "écrire des choses en FORTRAN" et, simultanément, j'ai constaté que si j'écrivais de tels programmes, c'était essentiellement pour les corriger, puis pour corriger les version corrigées, etc. (et non, comme on pourrait le croire, pour faire faire des choses par un ordinateur).

J'ai donc vécu de l'intérieur cette époque où, face à une tâche décrite de manière très floue, le réflexe était de sortir les feuilles spéciales destinées à recevoir les textes des programmes

FORTRAN, puis à courir aux perforatrices pour confectionner le paquet de cartes perforées correspondant, et plein de fébrilité, à soumettre à l'ordinateur le résultat de cette absence de réflexion préalable. Se passait alors ce qui doit immanquablement se passer en pareil cas : après les allers et retours de la perforatrice à l'ordinateur pour la correction des erreurs de syntaxe, le programme finissait par être compilé (= traduit) et, prenant le contrôle de l'ordinateur, par lui faire faire... tout autre chose que ce qui était attendu, cette autre chose étant le plus souvent rien du tout !

D'autres ont dépeint bien mieux que moi les résultats aberrants et les ravages de cette absence de méthode où programmation rimait avec connaissance de la syntaxe d'un langage et où ténacité et endurance tenaient lieu de réflexion et d'intelligence. (Cf. [3], [15], [51], ...).

Il ne faut pas confondre "dissertation française" et "accord du participe" !

Dans beaucoup de formations à la programmation, tout se passe souvent comme si l'on annonçait l'organisation de cours sur la dissertation française : le public intéressé s'inscrit en espérant qu'on explique comment on peut cerner un sujet, comment dresser un plan conducteur, comment argumenter, organiser sa pensée, ... Et il est tout surpris de trouver un cours sur l'accord du participe ou sur le pluriel des noms, quand ce n'est pas simplement une initiation à la dactylographie !

En programmation, il ne faut pas non plus se tromper de cible : programmer, c'est tout autre chose que maîtriser les commandes d'un ordinateur (fut-il micro) ou même qu'être un spécialiste de la syntaxe d'un langage de programmation ! Et la programmation est bien plus cousine de l'organisation des idées sous-jacente à l'art de la dissertation qu'à la connaissance des règles d'accord du participe.

Il ne faudrait pas non plus me faire dire ce que je ne dis pas : la connaissance des règles syntaxiques et orthographiques d'un langage de programmation est indispensable; sinon, jamais on ne pourra exprimer les marches à suivre conçues dans une version assimilable par l'ordinateur et, faute de la maîtrise de cette étape du "Comment dire ?", c'est tout le processus qui est hypothéqué. Mais ces connaissances "grammaticales", indispensables, sont notoirement insuffisantes pour programmer correctement.

4.2 Et les erreurs ?

Dans l'approche proposée ici, qui privilégie les concepts plutôt que les aspects techniques, la programmation peut paraître, à première vue, comme une activité relativement simple. La réalité montre chaque jour qu'il n'en est rien et que, fort souvent, toute cette démarche d'analyse débouche sur des programmes "incorrects" : l'exécutant-ordinateur ne traite pas la tâche de manière satisfaisante ! En vérité, **nous ne la lui faisons pas traiter correctement.**

> Je ne parle pas ici des erreurs de syntaxe qui peuvent subsister dans le texte du programme exprimé dans le langage de programmation. J'ai dit ci-dessus que ces erreurs empêchaient la traduction du programme, et, a fortiori son exécution. Mais ces erreurs (de forme) sont (le plus souvent) aisément corrigées et ne remettent en tout cas pas en cause le contenu de la marche à suivre elle-même : on peut avoir exprimé incorrectement des choses correctes.

Un très court exemple fera sans doute mieux saisir ces incohérences. Supposons un instant que la tâche à faire traiter par l'ordinateur consiste à "conjuguer un verbe régulier de la première conjugaison à l'indicatif présent". Sans préjuger des "capacités" de l'ordinateur et sans entrer dans une formalisation de la marche à suivre, on pourrait penser à des indications du genre de ce qui suit :

- demande quel est le verbe à conjuguer;
- retire les deux derniers caractères de ce qu'on t'a fourni (pour garder le radical);
- écris "je", puis passe un blanc, puis écris le radical auquel tu colles "e";
- en dessous, écris "tu", un blanc, puis le radical en y collant "es";
- puis "il", un blanc, le radical et "e";
- etc..

Remarquons qu'il s'agit de traitements purement formels et que, même si je n'ai encore rien dit de ce dont l'exécutant-ordinateur est capable, il s'agit là du type de manipulation dont il est friand. Ce n'est qu'une question de détails que de savoir comment lui exprimer ce qui précède à travers l'un ou l'autre langage compromis.

Une première "surprise", c'est qu'avec les explications fournies, il est parfaitement possible de faire "conjuguer" "plombier" ou même "valises". Sans sourciller et conformément à notre marche à suivre, l'ordinateur fournira :

je plombie	ou	je valise
tu plombies		tu valises
il plombie		...
...		

On rétorquera qu'il faut être de mauvaise foi pour demander la conjugaison de "verbes" comme ceux-là... et c'est tout à fait vrai, même si l'ordinateur ne sort pas "grandi" de telles facéties.

Encore une fois, l'ordinateur ne "sait" pas ce qu'est un verbe. Il ne peut s'attacher qu'à la forme de ce qu'il manipule en suivant consciencieusement nos indications.

Nous aurions pu nous prémunir contre un certain nombre de bêtises, en demandant par exemple à l'exécutant-ordinateur de vérifier que le "verbe" fourni se termine bien par "er" et sinon de le refuser (ce qui aurait évité la "conjugaison" de "valises", mais aurait laissé passer "plombier" ou "cerisier"). Mieux, nous aurions pu lui fournir une liste exhaustive de tous les verbes admis en lui demandant de n'accepter que les mots présents dans cette liste.

Remarquons cependant, qu'à chaque fois, c'est à nous, programmeurs, qu'il revient de prévoir les opérations supplémentaires qui vont permettre d'affiner le comportement de l'exécutant.

Revenant à un comportement plus habituel, nous aurons le plaisir de voir l'ordinateur conjuguer correctement les verbes "porter", "parler", "tirer", ... et nous décréterons donc notre programme correct, pour autant qu'on se limite réellement à des verbes réguliers du premier groupe.

Et cela jusqu'au jour où quelqu'un demandera la conjugaison du verbe "aimer", ce que l'exécutant-ordinateur, guidé par notre marche à suivre, fournira aussitôt sous la forme

je aime

tu aimes

il aime

...

ce qui montrera qu'un programme considéré comme valide ne l'est absolument pas !

Bien entendu, il ne s'agit là que d'un tout petit exemple, sans importance; il a cependant le mérite de mettre en évidence quelques vérités premières :

1. Il est vraiment difficile d'être exhaustif et de mettre en évidence TOUTES les règles qui régissent notre comportement, face à la réalisation d'une tâche, même très élémentaire. Quand je parlais, plus haut, de déplier complètement une tâche, sans rien laisser dans l'ombre et en faisant une chasse sévère aux implicites, c'est de cela que je parlais !

   C'est cette complexité qui fait de la programmation, quelle que soit la méthode employée, une activité compliquée : concevoir une marche à suivre pour faire effectuer une tâche par l'exécutant-ordinateur ne sera jamais simple et ne se réduit en tout cas pas à la connaissance des règles syntaxiques gouvernant le langage de programmation employé.

2. L'ordinateur est un exécutant rapide mais affreusement docile et obéissant. Il est généralement le dernier à mettre en cause lorsque le traitement d'une tâche ne s'effectue pas comme prévu. Ce qu'il faut revoir, presque toujours, c'est la marche à suivre qui lui est fournie et que NOUS avons conçue... en y laissant subsister des erreurs et des incohérences. Le drame, c'est souvent que **nous croyons lui dire autre chose que ce que nous lui disons vraiment!**

   C'est peut-être étonnant, mais il a fallu fort longtemps à l'informatique (ou plutôt aux informaticiens) pour se rendre compte qu'essayer un programme pouvait (avec de la chance) montrer qu'il était incorrect, jamais prouver qu'il était correct.

5. Exercices

Il est évidemment fort malaisé de proposer à ce stade des exercices "techniques", puisque ce chapitre introductif a surtout pour objet de baliser notre parcours futur au pays de la programmation.

1. Eclairez quelques facettes importantes des termes suivants :
   - Ordinateur
   - Programmer
2. Quelle est la tâche principale de l'analyste-programmeur ?
3. Donnez quelques exemples de traitements formels d'informations, puis des exemples de traitements non formels.
4. Pensez-vous que les problèmes liés à la commande numérique de certaines machines outils (tours par exemple) soient liés au monde du "faire faire" tel qu'il est présenté dans ce chapitre ?
5. Donnez quelques arguments qui prouvent que l'étape du "Quoi faire ?" est essentielle.



VOUZZAVEDIBISAR

Les Gaullicoquins et les Grapurinades

Edmond Bianco

Les chevaliers de l'Intelligence de la Gaullicoquie d'En-bas, montèrent une jacquerie qui menaçait de s'étendre à tout ce qui se comptait comme membre de ladite Intelligence. Alors le Premier Sinistre Grapurin, armé de sa plus belle voix grelottante de courroux fit une belle Grapurinade devant les étranges lucarnes. On le vit, tel l'Esprit d'Aladin bondissant hors de sa lampe magique, prêt à crever l'écran, la main droite contractée en grappin comme pour soulever des montagnes, et la main gauche virevoltant gracieusement autour de son minois verrouillé, tandis que son verbe roulé comme par un vieil acteur du Français, éructait avec force :

"...L'Intelligence de la main, ça existe aussi ... et en plus, elle passe par le cœur...". Cela voulut-il dire que les Intelligents d'en bas n'obtiendraient pas un sou de plus ? Mystère.

Pendant ce temps là, une immense catastrophe continuait à peser sur la Gaullicoquie. C'est pour le moins ce qu'en déclarait le Baron de la Cadière, chaque fois qu'un journaliste lui demandait son avis, ou bien qu'il rencontrait son ami Grapurin, c'est-à-dire souvent. En fait, les causes en étaient doubles, d'abord les trente cinq heures et aussi le trop de sécurité des travailleurs, qui, semblables à des fonctionnaires, prenaient leurs responsabilités trop à l'aise. Tout cela étant dramatiquement regrettable pour l'enrichissement du pays, et surtout de ceux qui contrôlaient cet enrichissement d'une main de fer dans un gant de velours. Il était visible que le gant de velours devenait de plus en plus gênant et superflu.

La Gaullicoquie allait mal, c'est sûr, de plus en plus mal. Et quand quelque chose va mal, il faut trouver des causes, et la cause là, crevait les yeux : les trente-cinq heures. Les hôpitaux marchaient mal (?) : les trente-cinq heures. les urgences étaient bloquées : les trente-cinq heures. Quinze mille morts pendant la canicule : les trente-cinq heures. Personne n'avait eu le temps d'avertir le Sinistre de la Santé, lui-même bien trop occupé pendant ses petites trente-cinq heures. Les caisses de l'état sont vides : les trente-cinq heures, la dette publique grimpe de manière dévastatrice (?) : les trente-cinq heures, etc... etc...

Mais le comble fut véritablement atteint lorsque le meilleur Gaullicoquin d'entre nous, toujours droit dans ses bottes se fit épingler par la justice, pris la main dans le sac. Jamais homme ne ramassa tant grande cueillette d'hommages. Il est vrai que toute la grande Satrapie au pouvoir lui devait fière chandelle. Non pas bouts de chandelles comme on aurait pu en attendre d'un gestionnaire économe, mais carrément grandes ripailles à la mesure de l'éclat dû au lustre, au panache, à la splendeur de cette grande équipe. Les juges impitoyables manquèrent de peu le pilori.

Des journalistes imprudents qui annoncèrent prématurément le retrait de la politique de ce digne ancien premier Sinistre, furent durement punis. Ah mais ! Et depuis, la Gaullicoquie d'en bas chante :

Il est tombé par terre, c'est la faute à Voltaire.
Le nez dans le ruisseau, c'est la faute à Rousseau.
Bousculé par un Beur, la faute aux trente-cinq heures...